Rapid distributed IPvX resolver

Authors:

- Alexey Shkittin
- Nikolay Labaznikov

Project Contributors:

- Alexey Shkittin
- Nikolay Labaznikov
- Semyon Loginov

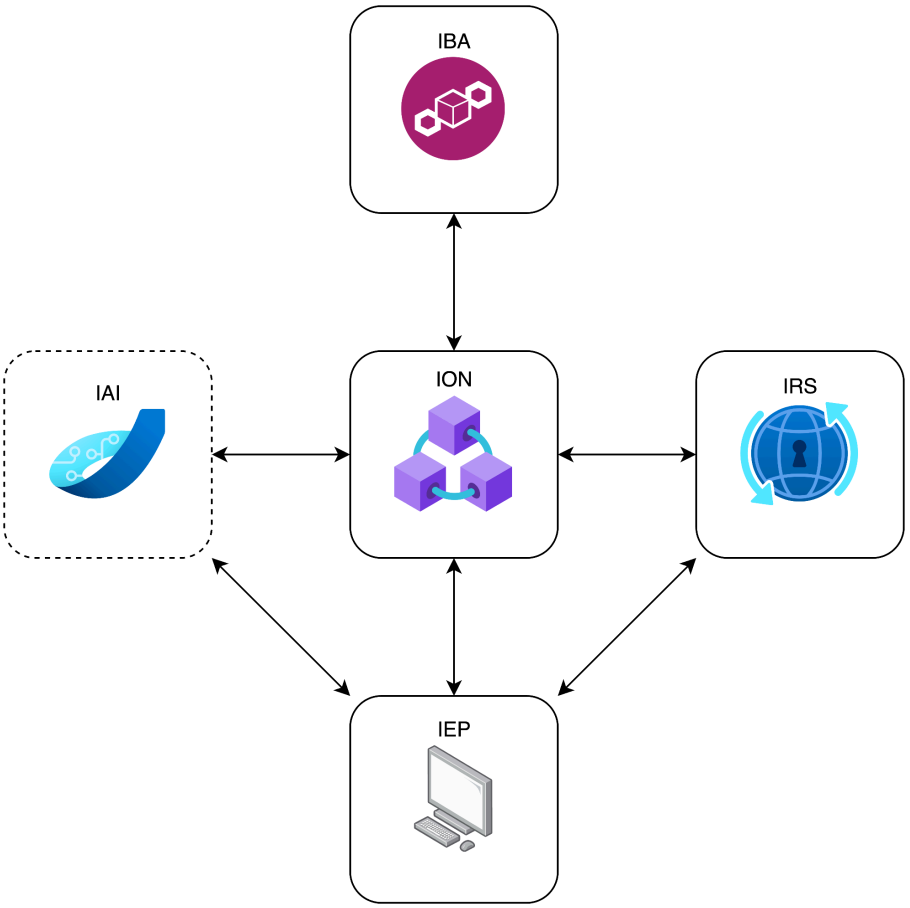
Preface

We had several publications where we have highlighted the problematic of the Domain Name System (DNS) - its drawbacks, limitations, and security concerns. [1] [2]

As our reader already got familiar with our IPvX Ecosystem, we just make a quick review, and proceed next - with the transition from DNS system to IPvX IRS system, where we move domain names databases to a “single but numerous” database in a form of a blockchain, and introduce a specially crafted resolver, capable of interacting with such database.

IPvX Structure

Our IPvX Ecosystem is a modular system, which architecture allows to change components regardless of a backend technology used. The blockchain should have an RPC available, the resolver should act like a real-world DNS resolver [3], and an intermediary, or “oracle” [4], should implement some API and be able to “talk” to the blockchain.



IPvX-adapted Blockchain (IBA)

Ethereum-compatible blockchain with Smart Contracts and Non-Fungible Tokens available. Represented by a series of EL+CL RPC-nodes [5]

IPvX Oracle Network (ION)

These are a network of Web-API nodes, which receive requests, apply logic to them, and communicate with IBA.

In the target scheme the nodes (or several groups of nodes) are utilized with anycast address (-es). [6]

IPvX Resolver System (IRS)

IPvX Resolver System is a distributed network of resolvers that provide the knowledge of the exact location of target node and target functionality of this node in IPvX Address Namespace.

Assume them as a black box, receiving DNS-requests and responding DNS-like, but not being a DNS resolver in fact.

IPvX Artificial Intelligence (IAI)

We leave space for this block to address situations, when a decision-making algorithm could not be pre-programmed. More of that, we assume, that a once mistakenly-programmed algorithm could be way worse than a guaranteed but unique answer in certain situations.

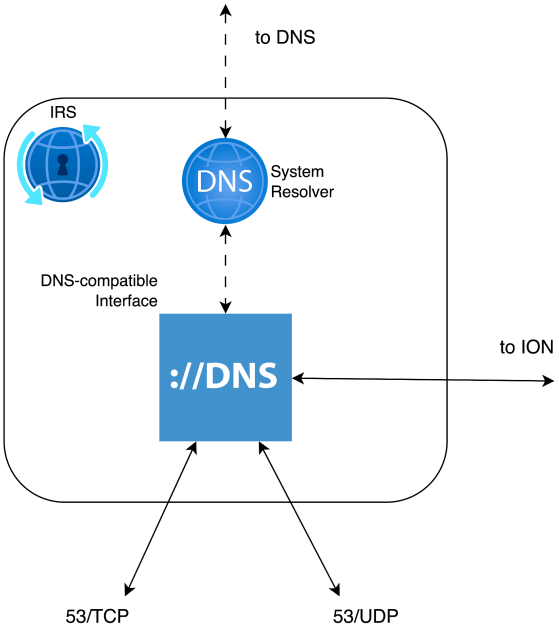
IPvX Endpoint (IEP)

This is the most numerous representative of the IPvX Ecosystem - user devices.

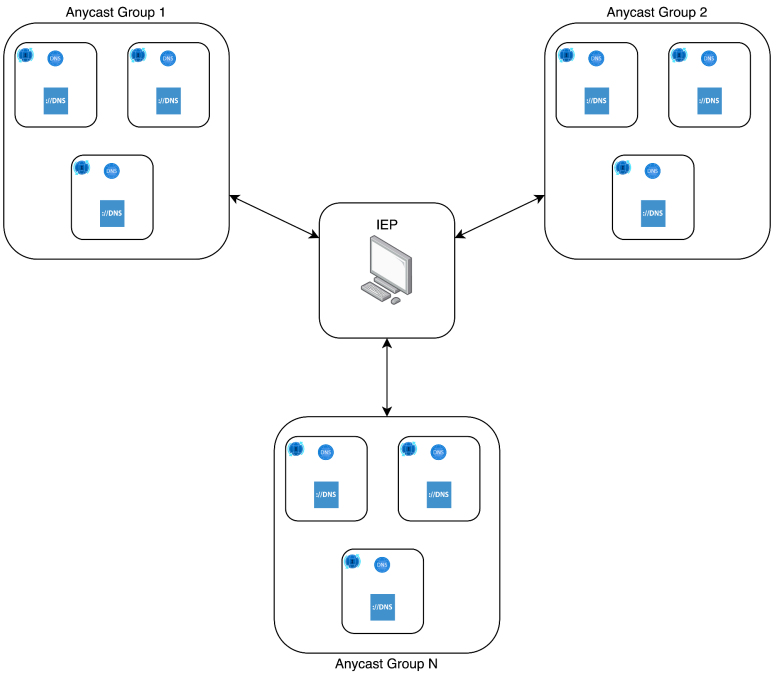
They can aim for a Root-Based DNS system as a default resolver, or IRS Anycast address if they want a rapid, precise, and guaranteed answer.

IRS Structure

IPvX Resolver System node consists of a DNS-compatible Interface, which listens to incoming connections on port 53 of protocols TCP and UDP like an ordinary DNS server does, and optionally a default system resolver, which is capable of performing regular DNS queries.

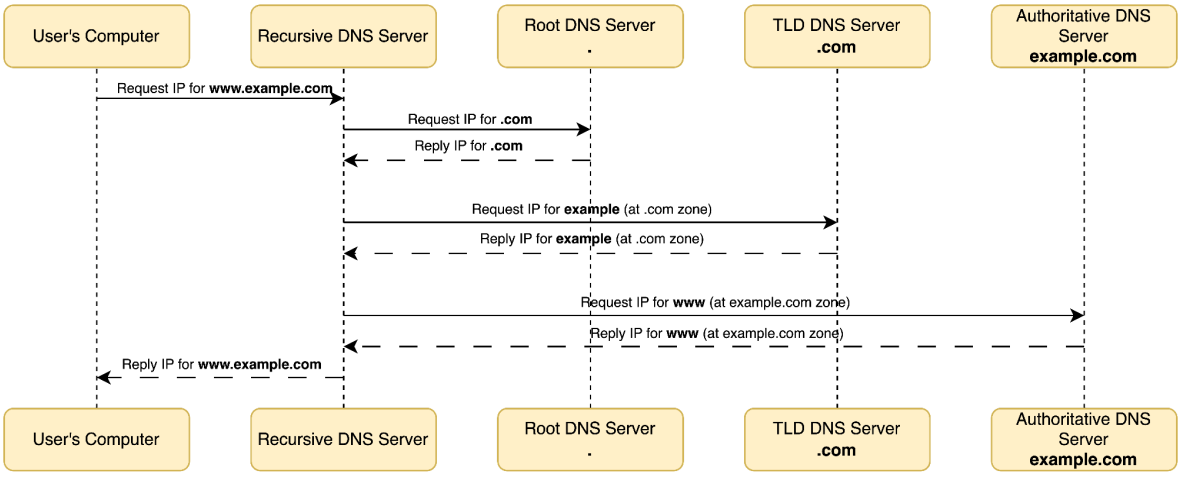


The target structure assumes there are lots of IRS nodes, and groups of them form publicly available anycast addresses.



DNS Interaction

Let’s remember how classic DNS requests and replies are performed.



When the user's device tries to reach “example.com”, the series of DNS queries take place, starting from your default resolver and then to the DNS Root and down to the target DNS zone server.

The reply which the user’s device receives is a DNS RRSET [7] .

RRSET (Resource Record Set) is a set of similar DNS resource records (RR), which form a record data and corresponding information.

Each RR consists of a name, class, type, TTL(Time to Live), and data.

This is a reply to a request to provide an address for the name “example.com”:

```
example.com.          1702  IN    A     93.184.215.14
```

Such DNS record is placed in .com zone and reads like:

```
“The name ‘example’ is at 93.184.215.14 and this record of class
‘Internet’ is valid for 1702 next seconds at the answered server”
```

```
$ dig @1.1.1.1 example.com A

; <<>> DiG 9.10.6 <<>> @1.1.1.1 example.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 648
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;example.com.                   IN      A

;; ANSWER SECTION:
example.com.            1702    IN      A       93.184.215.14

;; Query time: 42 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Jul 31 07:03:50 CEST 2024
;; MSG SIZE  rcvd: 56
```

The received reply is an RRSET, but consists of only one RR. We can find larger RRSETS:

```
$ dig @1.1.1.1 cnn.com

; <<>> DiG 9.10.6 <<>> @1.1.1.1 cnn.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7423
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;cnn.com.                       IN      A

;; ANSWER SECTION:
cnn.com.                46      IN      A       151.101.67.5
cnn.com.                46      IN      A       151.101.3.5
cnn.com.                46      IN      A       151.101.195.5
cnn.com.                46      IN      A       151.101.131.5

;; Query time: 41 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Jul 31 07:08:11 CEST 2024
;; MSG SIZE  rcvd: 100
```

This RRSET in reply consists of four RR: “cnn.com. 46 IN A” is a RRSET key part, and the addresses are a value (Resource Data) part. The result RRSET looks like:

```
cnn.com. 46      IN      A       151.101.67.5, 151.101.3.5, 151.101.195.5, 151.101.131.5
```

Let’s query the non-existent address and observe a reply:

```
$ dig @1.1.1.1 transition.ipvx.org

; <<>> DiG 9.10.6 <<>> @1.1.1.1 transition.ipvx.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22080
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;transition.ipvx.org.           IN      A

;; Query time: 193 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Jul 31 07:16:44 CEST 2024
;; MSG SIZE  rcvd: 48
```

The answer is:

```
transition.ipvx.org.  IN    A     <empty rdata>
```

Which reads as:

```
“the name ‘transition.ipvx.org.’ from the record of class
‘Internet’ and of type ‘Address’ is nowhere”
```

NB: observe the trailing dot after names in replies - this is a correct recording of a DNS name, because domains of every level are separated with a dot, and it assumes there is also a root zone (like “transition.ipvx.org.root”), but the “root” name is omitted.

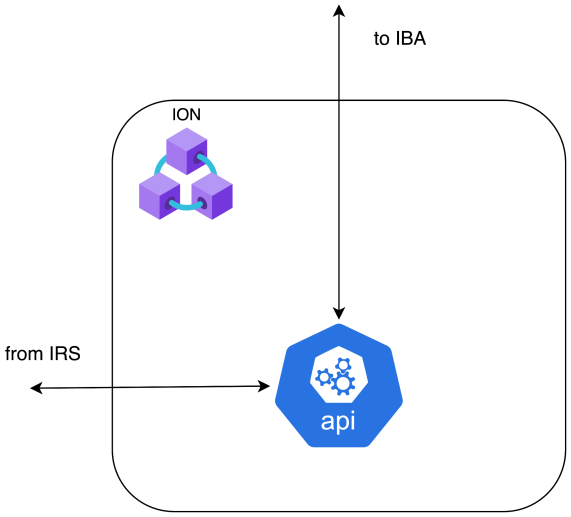
IPvX Interaction

Here parts of IPvX come together, and we will deepen into the details of how data flows, which requests and in which form are executed, and all stuff needed to get familiar with the proposed resolving process.

ION

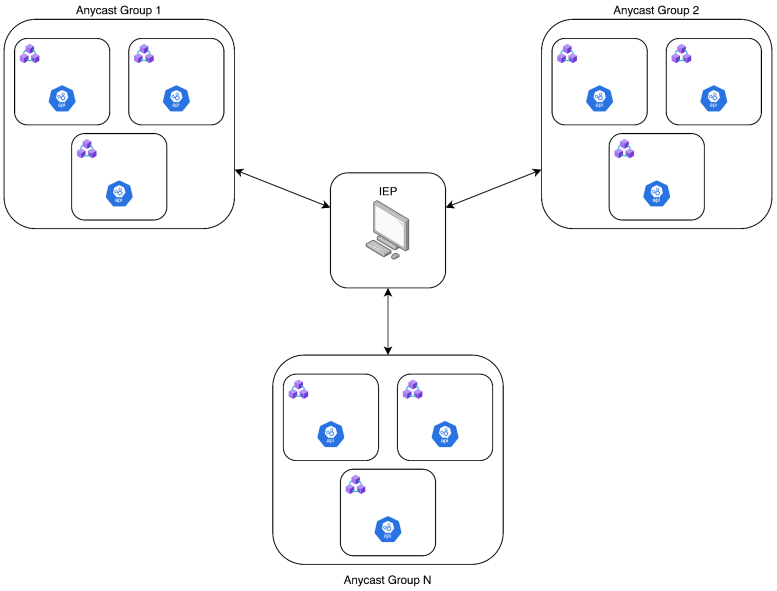
These nodes serve as a bridge between the outer world data and blockchain data.

IRS nodes address requests to ION API, ION transforms these requests address them to IBA, its Smart Contracts and NFTs.



The connection to IBA is made through the RPC (Remote Procedure Call) endpoint available at nodes.

And again, consider lots of ION nodes, and groups of them form publicly available anycast addresses.



Smart Contracts

The Ethereum Blockchain is named an “Ethereum Virtual Machine” (EVM) [8] meaning that it is not solely a blockchain, but also a distributed computing platform, capable of running logic and applications.

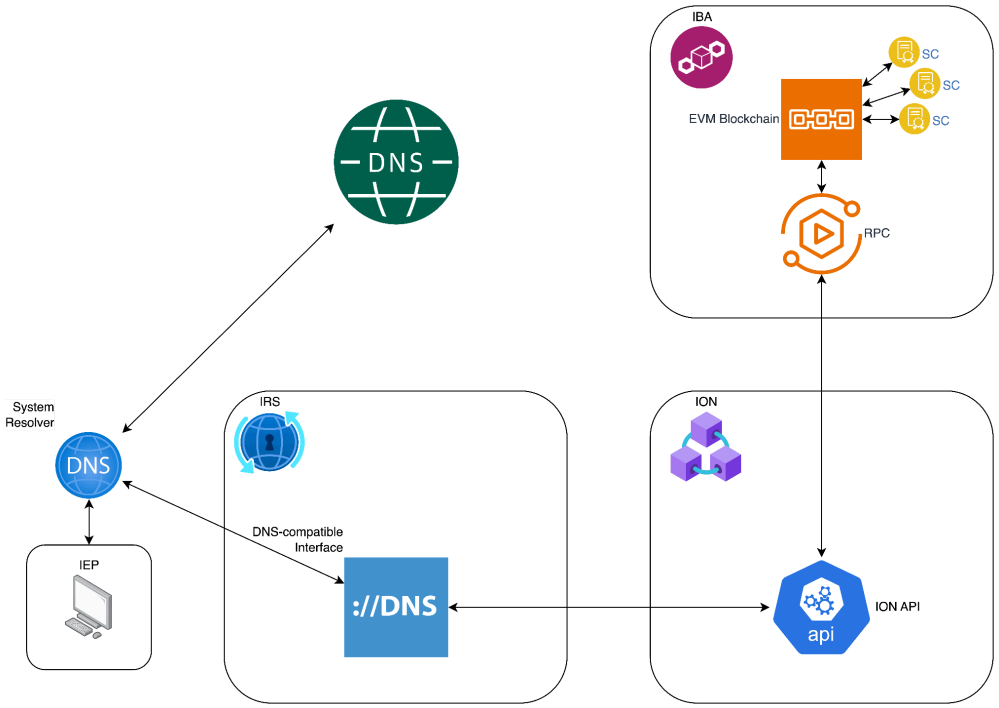
Our ION system (and IRS by means of ION) utilizes this possibility by interacting with Smart Contracts (SC), containing the logic of dealing with DNS records, which in this case we call “IRS Records” because of several differences.

Interaction with a SC means performing a transaction or a call by a certain blockchain address, at which the SC is located.

IRS

The IRS system does not interact with the IBA directly, but uses the ION API. Let’s combine all the schemes together. But jumping a bit forward, we have two great opportunities here on how to utilize classic DNS alongside with the new IRS.

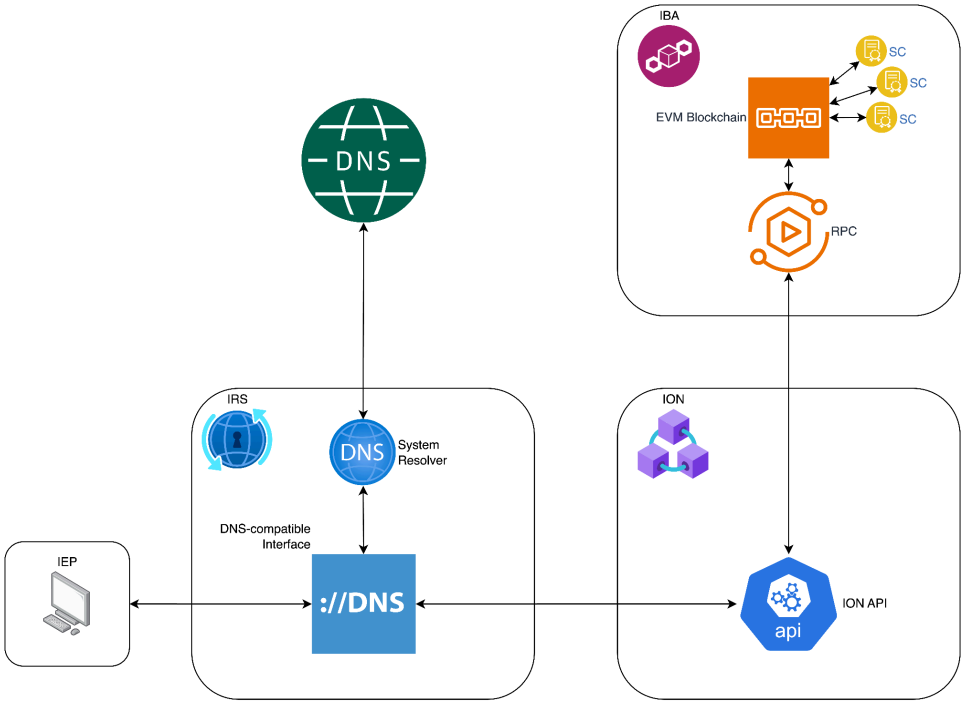
The first scheme represents a common approach. The endpoint performs a request to its own system resolver, which first addresses DNS Root, which in turn replies with the information where to find the address of the target name.



The System Resolver finally gets to the IRS custom resolver. IRS DNS-like interface receives a request, and sends it to ION API.

ION transforms a request into a SC form and calls the SC with parameters of the initial DNS request (queries IRS RRSET). If the target name is found under the called SC, the IRS receives a reply with a DNS RRSET, and transmits it to the caller (IEP system resolver).

Another approach can take place, when the IEP uses not the default system resolver, but a manually added IRS address at this point.



IRS first queries ION (and IBA by its means) for a target name, and if it hasn't been found - queries its own default system resolver, which addresses the initial request to Root DNS.

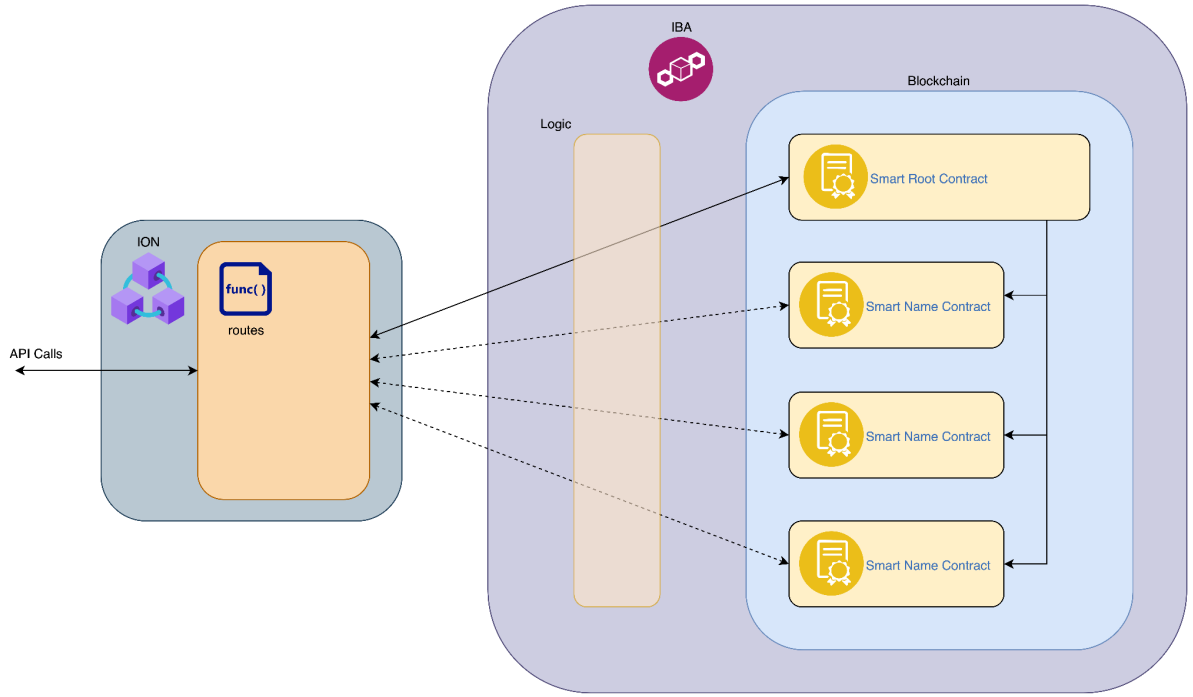
This approach allows to shorten overall reply time for the records, contained in IBA, but overloads IRS system resolver with non-relevant requests, when a target name is not of a IBA-contained.

Along with the shortened IBA-contained replies, another opportunity could be taken on account - the possibility of using DNS records of non-Root DNS namespace. For example, to use the zones, which do not exist in the DNS system, therefore omitting DNS Root from the process of government for such zones. But with this opportunity other challenges come, and we will cover them later.

The combination of the two schemes is to set IRS as a primary resolver for an IEP, but make IRS not to answer for non-relative zones. Thus, IBA will first query the IRS, and IRS will reply with an IBA-hosted record. If the record is not found, IRS will not reply, and the IEP will query the next alternative configured resolver.

ION Connection

ION is serving a web api with routes available to address IBA smart-contracts. Each of the routes addresses smart-contract external logic (to define the parameters required to call a smart-contract).



These routes are:

/create_dns_zone

- Addresses main SC: passes zone name, owner's address
- The result is a dns zone SC created, the owner is set, and the address of SC replied

/get_dns_zone

- Replies with an owner and SC address if any

/set_dns_zone_owner

- Takes current owner, the digital signature, the new owner
- Sets the new owner of a Smart Name Contract

/get_dns_zone_owner

- Returns an owner of a Smart Name Contract

/set_dns_name_record

- Addresses Smart Name Contract
- Takes DNS RRSET Record
- Outputs IRS RRSET Record
- Transacts IRS RRSET to Smart Name Contract

/get_dns_name_record

- Returns IRS RRSET

/delete_dns_name_record

- Sets IRS RRSET value to empty

IRS RRSET Theory

Let's get to know what an IRS RRSET is.

It is DNS RRSET alike, with the difference in division into key part and value part.

As we already know, the uniqueness of a DNS RRSET is defined by the RR name, type, and TTL. We want to transform each DNS RRSET into a form, suitable to be stored on a blockchain.

To save space and to minimize calculation difficulty during resolving, we store RRSETs in a form of a dictionary { "key": "value" }, where the "key" part is unique-makers of a RRSET, and the "value" part is a value itself.

So, the record:

```
example.com.          1702  IN    A     93.184.215.14
```

will be represented as:

```
{"example.,A": "1702,93.184.215.14}
```

We omit "IN" class as we use the only one, and the gone "com." is the name of the used Smart Name Contract, so we have no reason to overload a record with it.

Before being stored in IBA, each part is represented in bytes. Then this record is stored as a mapping in a Smart Name Contract for this zone.

Mappings allow users to rapidly access records in blockchains in a guaranteed interval of time. The cost you pay - you have to know the record "key" part in advance, which means you cannot list all available records (for ex.), but you can access a single record by its name. This is made by storing records in a form of hashes: the "key" part hash represents the "value" part, so you cannot query "value" without knowing the "key" hash.

We assume that the zone owner possesses all own records, and precisely knows which records they have stored, and other internet users have no business of the complete zone record set (just like in a traditional DNS world, the complete zone file is a secured information).

We can observe IRS-ION interaction if node logs.

- ipvx-irs - is an IRS machine
- ipvx-ion - is an ION machine

```
$ dig ipvx.org
```

```
ipvx-irs  | [irs_api.py:0072] [20240811-005906] Try resolve .ipvx.org.
ipvx-irs  | [irs_api.py:0084] [20240811-005906]         Request: {'dns_name': 'ipvx.org', 'name': '.', 'type': 'A'}
ipvx-ion  | [ion_api.py:0324] [20240811-005906] Request to get RRSET:
ipvx-ion  | [ion_api.py:0325] [20240811-005906]         Zone: ipvx.org
ipvx-ion  | [ion_api.py:0326] [20240811-005906]         Name: .
ipvx-ion  | [ion_api.py:0327] [20240811-005906]         Type: A
ipvx-ion  | [ion_api.py:0354] [20240811-005906] OK
ipvx-ion  | [ion_api.py:0355] [20240811-005906]         IRS RRSET: {'.,A': '60,195.133.0.77'}
ipvx-ion  | 172.19.0.3 - - [11/Aug/2024 00:59:06] "POST /get_dns_name_record HTTP/1.1" 200 -
ipvx-irs  | [irs_api.py:0099] [20240811-005906]         Reply: {'.,A': '60,195.133.0.77'}
ipvx-irs  | [irs_api.py:0103] [20240811-005906] OK
ipvx-irs  | [irs_api.py:0180] [20240811-005906]         ip_address: 195.133.0.77
```

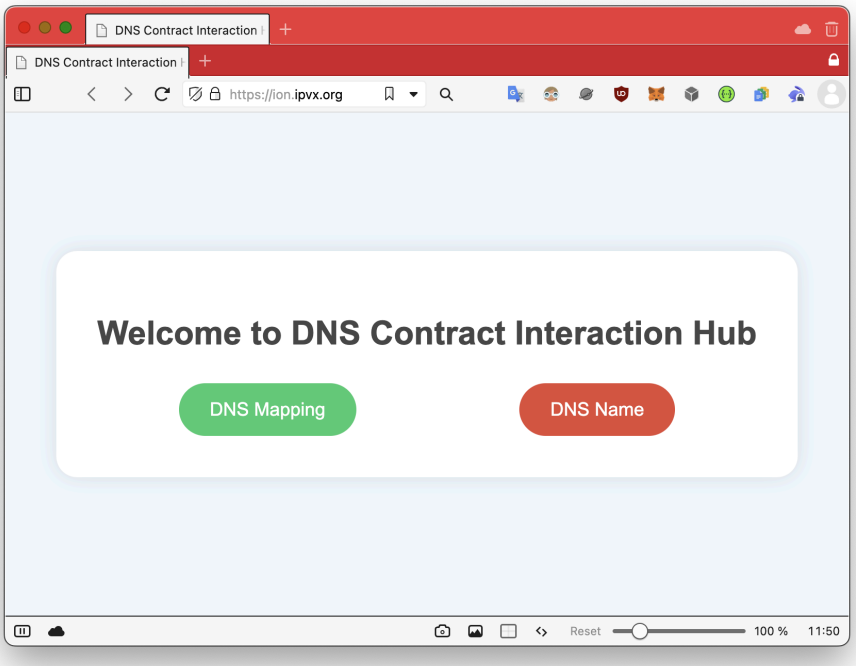
- IRS receives a regular DNS request to query ipvx.org A type RRSET
- IRS passes the request to ION
- ION converts the request to a form of an IRS RRSET and asks IBAif there is such record(if there is a “key” part in a zone SC records)
- IBA (not shown) returns IRS RRSET
- IRS receives an IRS RRSET, and constructs a regular DNS reply with the “value” part
- The client receives a DNS reply:

```
$ dig @212.192.222.75 ipvx.org

; <<>> DiG 9.10.6 <<>> @212.192.222.75 ipvx.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18902
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;ipvx.org.                      IN      A

;; ANSWER SECTION:
ipvx.org.               60      IN      A       195.133.0.77

;; Query time: 79 msec
;; SERVER: 212.192.222.75#53(212.192.222.75)
;; WHEN: Sun Aug 11 02:59:06 CEST 2024
;; MSG SIZE  rcvd: 53
```

Smart Contract Hub

The ION API is publicly accessible, and there is also a web interface available for interaction.

These are simple web pages with demonstrations of the concept of dealing with blockchain-hosted DNS zones and records.



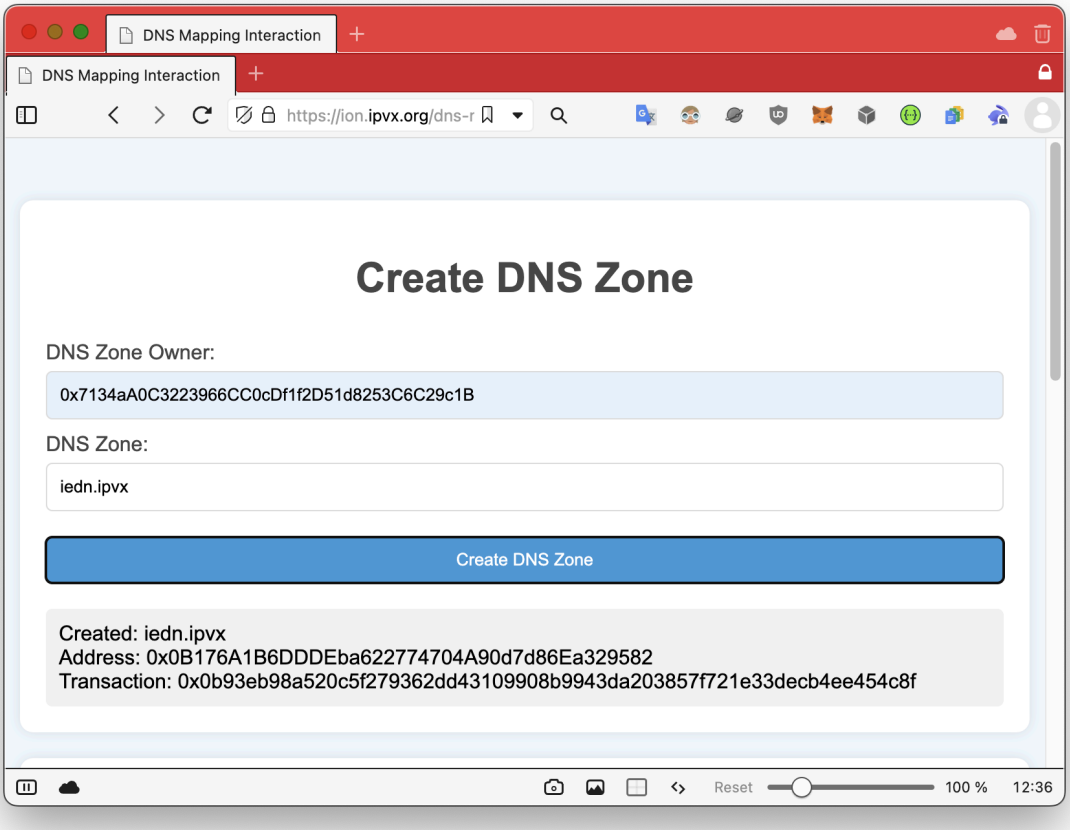
The start page prompts us to choose whether we want to create a DNS zone (“DNS Mapping") or to create records for our previously created and owned DNS zone.

Well-known DNS names

We start with the creation of a zone. We start with an example when we own a ICANN DNS Root compatible domain name - “ipvx.org”. The domain is just bought from some Domain Names Registrator, and there is no DNS or Web hosting assigned to it.

We will use our IPvX IRS system to create a DNS server and to host DNS records (ok, let’s say “create a DNS server-like entity”).

Set an owner, a “zone” name, and proceed:

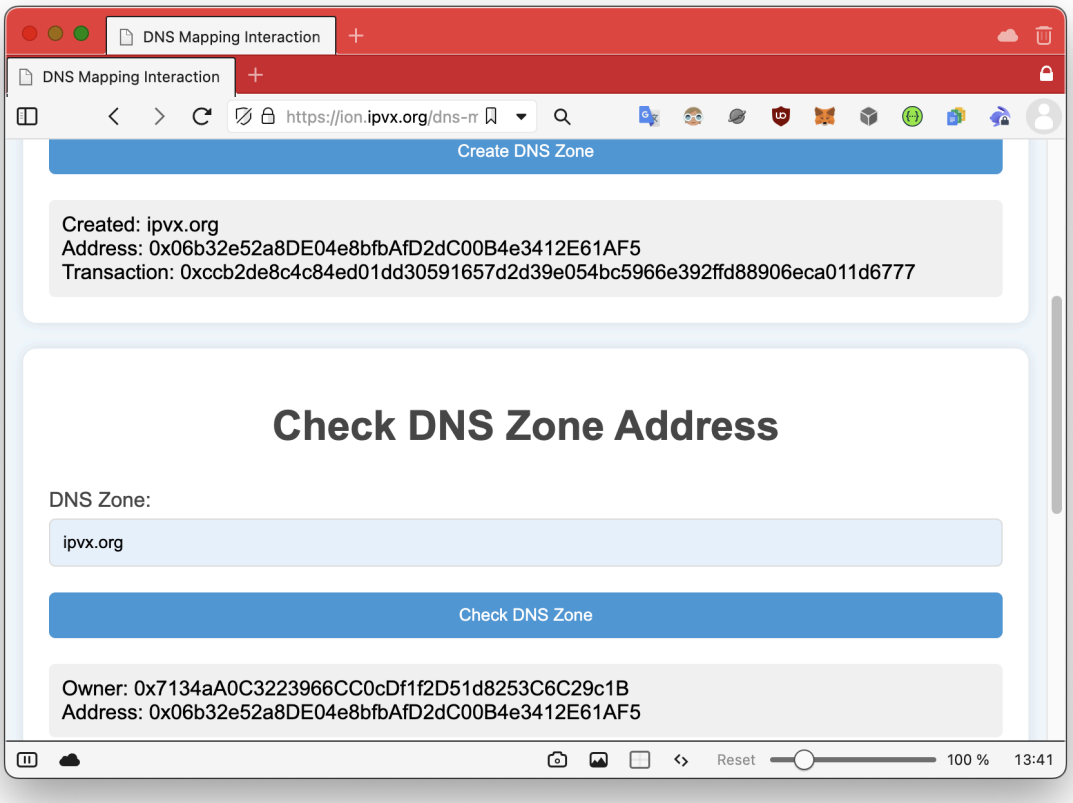


Before the zone creation, some verifications occur to check whether the creation of a zone with such name is technically legal. For example, if a zone with such name already exists within the global DNS system, or maybe if such domain has already been registered, but not delegated yet.

It replies with a successful zone creation, the address of the Smart Name Contract, and the transaction address. Check the logs:

```
ipvx-ion  | [ion_api.py:0168] [20240811-113643] Request to create DNS Zone:
ipvx-ion  | [ion_api.py:0169] [20240811-113643]         Zone: ipvx.org
ipvx-ion  | [ion_api.py:0170] [20240811-113643]         Owner: 0x7134aA0C3223966CC0cDf1f2D51d8253C6C29c1B
ipvx-ion  | [ion_api.py:0193] [20240811-113651] OK
```

and hub page:

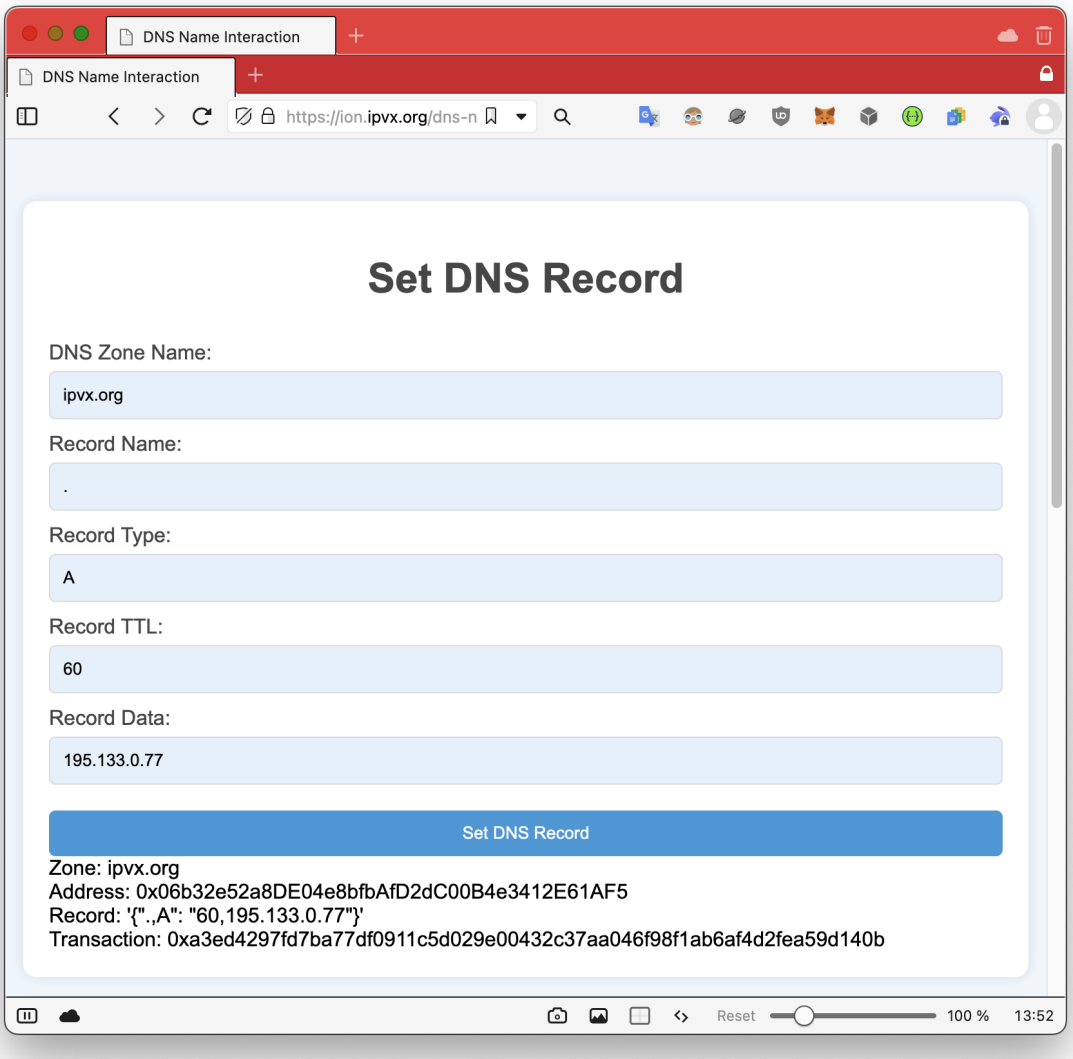


Seems to be OK. Now we add DNS records to our zone. This set has to be defined, as we want a minimum of a website at “ipvx.org” address, and name servers addresses:

.	60	IN	A	195.133.0.77
ns1.ipvx.org.	60	IN	A	212.192.222.75
ns2.ipvx.org.	60	IN	A	195.133.0.78

Here the first magic comes into play, as we do not need to have a dedicated DNS server for our zone - the blockchain and entrypoint (RPC) addresses would serve for that!

Let’s break down into one record creation example, others would look similar. We go to Smart Name Contract part of the Hub (under “DNS Name” button), fill the fields and get records persistent within IRS system:



The output reads as follows:

> The zone “ipvx.org” hosted at address “0x06b32e52a8DE04e8bfbAfD2dC00B4e3412E61AF5” contains IRS RRSET '{".,A": "60,195.133.0.77"}' set by transaction “0xa3ed…”

which is the equivalent of a DNS record we need. And the ION log is:

```
ipvx-ion  | [ion_api.py:0255] [20240811-121550] Request to set RRSET:
ipvx-ion  | [ion_api.py:0256] [20240811-121550]         Zone: ipvx.org
ipvx-ion  | [ion_api.py:0257] [20240811-121550]         Name: .
ipvx-ion  | [ion_api.py:0258] [20240811-121550]         Type: A
ipvx-ion  | [ion_api.py:0259] [20240811-121550]         TTL: 60
ipvx-ion  | [ion_api.py:0260] [20240811-121550]         Data: 195.133.0.77
ipvx-ion  | [irs_records.py:0091] [20240811-121550] Attempting to create record type: A with data: 195.133.0.77
ipvx-ion  | [irs_records.py:0097] [20240811-121550]     Record class: <class 'dns.rdtypes.IN.A.A'>
ipvx-ion  | [irs_records.py:0100] [20240811-121550]     Processed data: 195.133.0.77
ipvx-ion  | [irs_records.py:0242] [20240811-121550]     RRSET: . 60 IN A 195.133.0.77
ipvx-ion  | [ion_api.py:0267] [20240811-121550] Preparing to post IRS RRSET: {'.,A': '60,195.133.0.77'}
ipvx-ion  | [ion_api.py:0298] [20240811-121615] OK
ipvx-ion  | [ion_api.py:0299] [20240811-121615]         IRS RRSET: {'.,A': '60,195.133.0.77'}
ipvx-ion  | 172.19.0.1 - - [11/Aug/2024 12:16:15] "POST /set_dns_name_record HTTP/1.0" 200 -
```

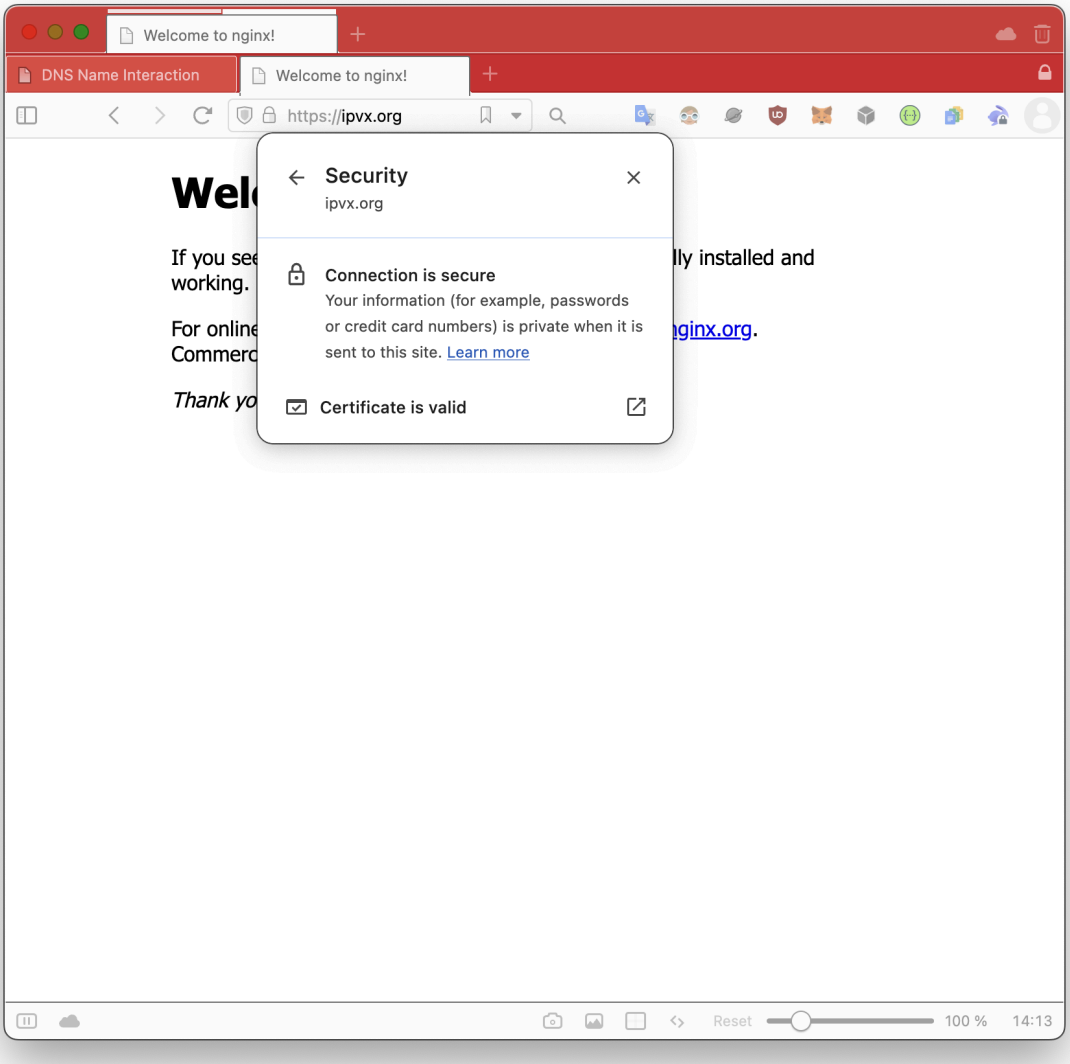
Now, when we set our three records, we ask the Domain Name Registrator to set ".org" zone records, pointing to name servers of our zone. That would be the set:

ipvx.	60	IN	NS	ns1.ipvx.
ipvx.	60	IN	NS	ns2.ipvx.
ns1.ipvx.	60	IN	A	212.192.222.75
ns2.ipvx.	60	IN	A	195.133.0.78

With this done, our zone is publicly accessible on the Internet, and is ready to host a website. Let's check:

```
2024-08-11 14:07:42 ~
$ dig ipvx.org NS +short
ns2.ipvx.org.
ns1.ipvx.org.
2024-08-11 14:08:08 ~
$ dig ns1.ipvx.org +short
212.192.222.75
2024-08-11 14:08:39 ~
$ dig ns2.ipvx.org +short
195.133.0.78
2024-08-11 14:08:42 ~
$ dig ipvx.org +short
195.133.0.77
```

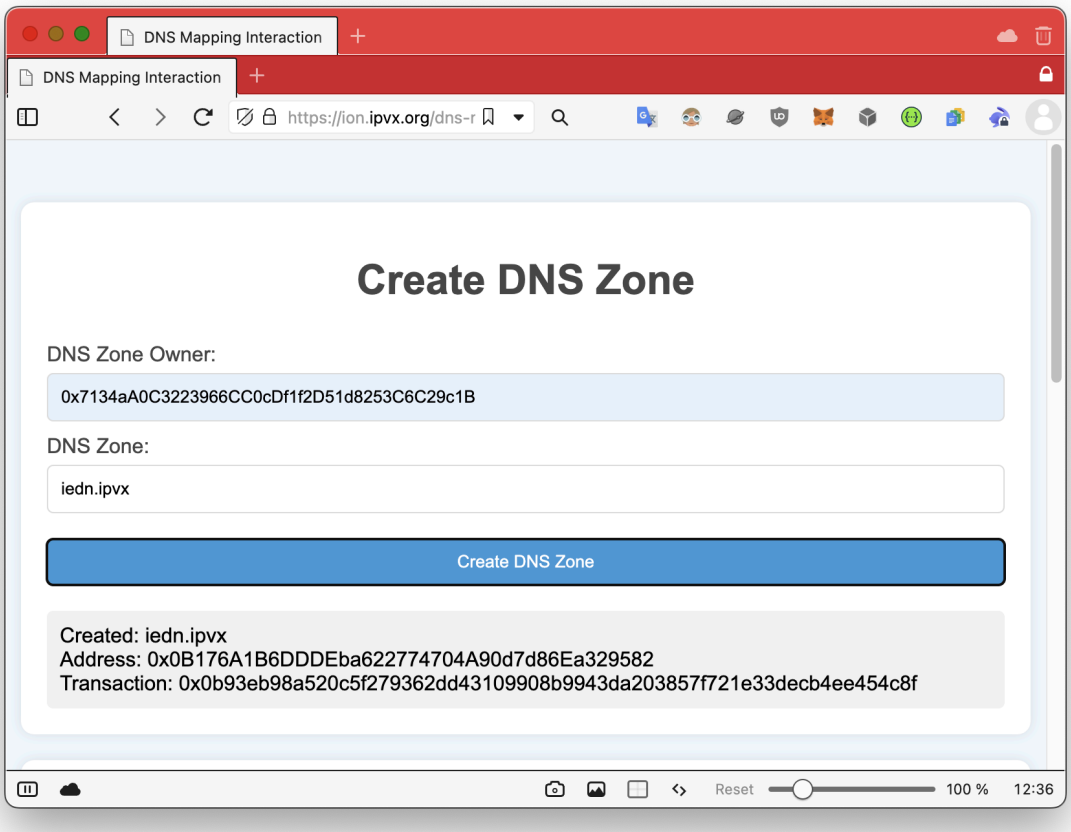
So, we create a nginx-powered website, and apply for a Letsencrypt Certificate, as our site is of ICANN DNS Root (every Letsencrypt check is passed during certificate request).



Non-ICANN DNS names

Here the process is going to be more complicated. First, we check whether the zone name is technically legal, as we made during “ipvx.org” IBA registration. Next, we check if a new name interferes somehow with some already IBA-hosted name. These two procedures have rather complicated logic that we will cover in next publications, and here we focus on the resolving process.

Let’s create a zone with the name “iedn.ipvx” - that would be a “top-level domain” which would not exist in ICANN DNS Root.



We set the owner of the new zone, the zone name, click “Create DNS Zone” and it is all done. The transaction is signed with a Smart Root Contract owner - we conceal these background operations as being out of scope of this article, but will cover them in later publications.

```
ipvx-ion  | [ion_api.py:0168] [20240811-212517] Request to create DNS Zone:
ipvx-ion  | [ion_api.py:0169] [20240811-212517]         Zone: iedn.ipvx
ipvx-ion  | [ion_api.py:0170] [20240811-212517]         Owner: 0x7134aA0C3223966CC0cDf1f2D51d8253C6C29c1B
ipvx-ion  | [ion_api.py:0193] [20240811-212527] OK
ipvx-ion  | [ion_api.py:0194] [20240811-212527]         Address: dns_name_contract_address
ipvx-ion  | 172.19.0.1 - - [11/Aug/2024 21:25:27] "POST /create_dns_zone HTTP/1.0" 200 -
```

OK, the transaction seems to occur in ION logs, and additionally we see the result of creation at a web-page.

Next, we check our created zone on a Smart Root Contract page, and it is also OK.

Now we go into Smart Name Contract, following “DNS Name” from the start page.

Here we create zone records, following the above described conceptions on DNS RRSETs and IRS RRSETs, and the resolving process.

This set of DNS Records has to be done:

.	60	IN	NS	ns1.iedn.ipvx.,ns2.iedn.ipvx.

```
ns1.        60    IN    A    212.192.222.75
ns2.        60    IN    A    195.133.0.78
```

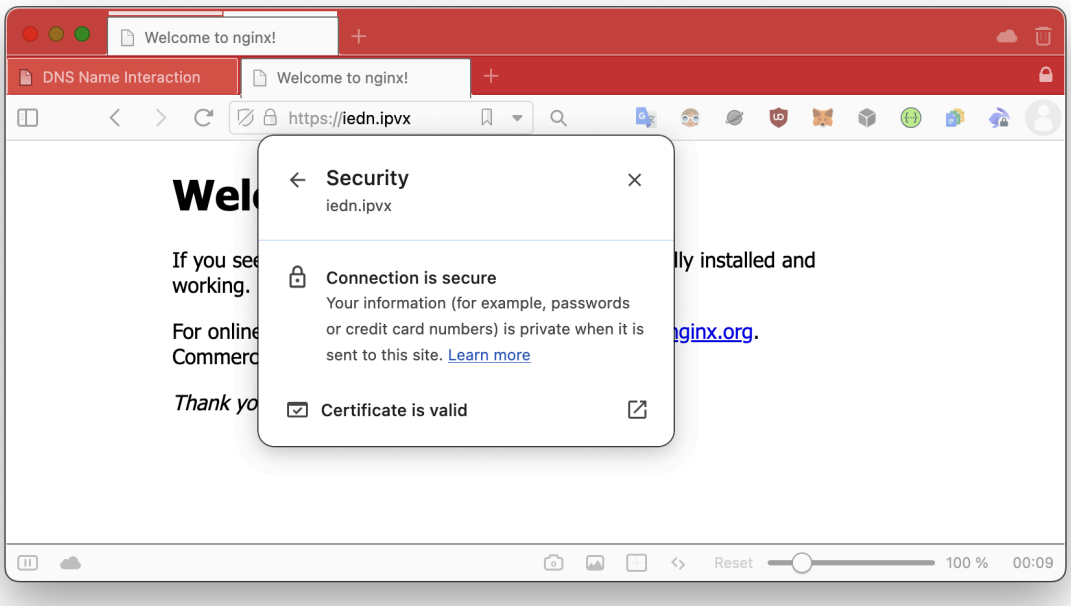
and we add an address record for our future web-server:

```
.           60    IN    A    195.133.0.76
```

The result is our web server address resolves correctly:

```
2024-08-11 23:52:44 ~
$ dig iedn.ipvx NS +short
ns1.iedn.ipvx.
ns2.iedn.ipvx.
2024-08-11 23:53:02 ~
$ dig ns1.iedn.ipvx +short
212.192.222.75
2024-08-11 23:53:26 ~
$ dig ns2.iedn.ipvx +short
195.133.0.78
2024-08-11 23:53:30 ~
$ dig iedn.ipvx +short
195.133.0.76
```

Now our name is prepared to host a web-site:



You may observe that “iedn.ipvx” is resolved through system resolver, meaning the resolved name is available not only in the browser, but in any application, and that our web-site has a

certificate, treated by the browser as a valid one. Consider this as a second magic, but actually several obvious solutions are lying behind. We plan to discuss the establishment of trust relationships in future publications.

Problems

The main problem we expect is the regulations process for names assignment. Shall the dot character be treated as a zone delimiter and be restricted in names? Shall every name be treated as a top-level domain without any subordinate relationships or we need to retain the possibility of hierarchy?

This is subject to discussion, though we have a vision of solving this problem.

How IPvX IRS addresses DNS issues

There are known DNS limitations and drawbacks:

Centralized control

Each zone is capable of running independently, and as you are an administrator of the zone, it is capable to run trustworthily to the zone parties.

Problems begin when you follow hierarchy and establish interconnections with neighbor zones.

The point of trust moves to the superior zone, until root. That’s why you cannot run your zone independently any more.

At last, each of the domains relies on DNS root operability.

IPvX IRS System provides the possibility to define numerous root “zones” (that are not actually zones any more) and to treat your IRS name as self-sufficient and independent.

And for conflicts resolving there is always a voting mechanism available with blockchain.

Naming limitations

Your domain and corresponding website meet limitations on its naming.

It is mandatory to have all of the superior zones named trailers, separated by dots.

From security considerations there came conventional rules for domain namings, individual for a given zone. For ex., you cannot mix symbols from different layouts, or you cannot use some special symbols.

Though it is still a subject for discussion, in IPvX IRS we see loose restrictions on naming conventions.

Reactive to changes

From scalability limitations, there came a DNS cache extension.

When you query a website, having several zones in domain name, your DNS resolver queries each zone DNS server, beginning from root level. Obviously, top-level servers, especially root zone, could not bear such load for the whole Internet.

DNS caching was introduced, when you define a lifetime for your DNS record. And servers across the Internet remember it.

When you make changes, there is a possibility that there can be some DNS server on the Internet that still remembers your old record, until caching time expires.

With IPvX IBA the name information is immediately available at any blockchain node, because the distributed resolver is synced in a short time, and there is no need to set huge TTLs. However it is a subject for discussion in terms of the distributed resolver overload that may occur with numerous records and short TTLs.

No security by design

Initially, the network was small, and every party relied on others. Nobody tried to make attacks on the DNS system.

With the network growth there had to be changes done to DNS protocols, but security issues had not been addressed. Scalability performed only in terms of availability.

Being blockchain-enabled the IPvX IRS System offers possibilities to implement cryptography controls by design, and we work on the implementation, but it is tough to consider backwards compatibility with traditional DNS systems.

No resource identification

There is no resource identification in the DNS system.

You can only be sure that a given domain name belongs to a given IP address. But you can not identify the owner of a resource or the querying party.

DNSSEC security extension was introduced. But it addresses integrity only, and not authenticity or non-repudiation.

For IPvX, with blockchain addresses assigned to any blockchain entity, it is possible to identify the IRS counterparties.

Conclusion

We can summarize that a blockchain-enabled Internet name resolving system is possible. New challenges arise and new problems are to be solved, especially related to compatibility with traditional DNS systems, the blockchain design and security problems, naming conventions, and governance.

References

1	A. Shkittin, N. Labaznikov “Internet Governance: From traditional to IPvX Ecosystem”	https://archive.org/details/ipv-x-ecosystem_202405
2	IPvX Network Team “Transitioning from DNS to IPvX with NFT Identifiers”	https://www.linkedin.com/feed/update/urn:li:activity:7205498999464726528
3	Wikipedia “Domain Name System”	https://en.wikipedia.org/wiki/Domain_Name_System
4	Wikipedia “Blockchain Oracle”	https://en.wikipedia.org/wiki/Blockchain_oracle
5	Alchemy “What is an RPC node?”	https://www.alchemy.com/overviews/rpc-node
6	Cloudflare “What is Anycast? | How does Anycast work?”	https://www.cloudflare.com/pl-pl/learning/cdn/glossary/anycast-network
7	Readthedocs “Retrieving and Creating DNS Records”	https://desec.readthedocs.io/en/latest/dns/rrsets.html
8	Ethereum “ETHEREUM VIRTUAL MACHINE (EVM)”	https://ethereum.org/en/developers/docs/evm